Memory management for lock-free data structures : an exercise

The following is the enqueue method from the lock-free queue of Michael and Scot (see https://www.research.ibm.com/people/m/michael/podc-1996.pdf)

```
enqueue(Q: pointer to queue_t, value: data type)
    node = new_node()                                  # Allocate a new node from the free list
    node–>value = value                                # Copy enqueued value into node
    node–>next.ptr = NULL                              # Set next pointer of node to NULL
    loop                                               # Keep trying until Enqueue is done
        tail = Q–>Tail                                 # Read Tail.ptr and Tail.count together
        next = tail.ptr–>next                          # Read next ptr and count fields together
        if tail == Q–>Tail                             # Are tail and next consistent?
            if next.ptr == NULL                        # Was Tail pointing to the last node?
                if CAS(&tail.ptr–>next, next, <node, next.count+1>)  # Try to link node at the end of the linked list
                    break                              # Enqueue is done. Exit loop
                endif
            else                                       # Tail was not pointing to the last node
                CAS(&Q–>Tail, tail, <next.ptr, tail.count+1>)        # Try to swing Tail to the next node
            endif
        endif
    endloop
    CAS(&Q–>Tail, tail, <node, tail.count+1>)          # Enqueue is done. Try to swing Tail to the inserted node
```

In this exercise we shortly discuss the challenges in creating memory management for the queue. The enqueue method is the more challenging to work with among the queue methods.

A. Epochs

1. Explain what should be added the code to make sure that the memory reclamation works well.

B. Hazard Pointers 1. Explain where one needs to protect a pointer with hazard pointers2. Describe a setting in which reclaiming without using hazard pointers will create a problematic race.

C. Optimistic Access

1. Propose a memory reclamation that protects writes to the data structure with hazard pointers, but does not use hazard pointers to protect read values. Instead, uses optimistic access to validate the read values.